

PATENT
5181-87600
Sun P6070

I hereby certify that this correspondence, including the attachments, is being deposited with the United States Postal Service, Express Mail – Post Office to Addressee, Receipt No. EL849601767US, in an envelope addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231, on the date shown below.

February 28, 2002

Date of Mailing


Shayna Blackmar

ARBITRATION SCHEME FOR EFFICIENT PARALLEL PROCESSING

Invented by:

Mark E. Pascual

Michael G. Lavelle

Nandini Ramani

Patrick Shehane

Attorney Docket No.: 5181-87600

Jeffrey C. Hood /JWC
Conley, Rose & Tayon, P.C.
P.O. Box 398
Austin, TX 78767-0398
Ph: (512) 476-1400

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates generally to the field of computer graphics and, more particularly, to high performance computer graphics systems.

Description of the Related Art

Early graphics systems were limited to two-dimensional (2D) graphics, a gray scale value for each pixel displayed, and acted as simple translators or interfaces to a display device. Modern high performance graphics systems, however, support three-dimensional (3D) graphics with textures and many special effects such as anti-aliasing, texture mapping, advanced shading, fogging, alpha-blending, and specular highlighting. 3D graphics data may be several orders of magnitude larger than comparable 2D graphics data. 3D graphics data may include many information components for each vertex of the geometric primitives used to model the objects to be imaged. These vertex information components may comprise: x, y, and z position; normal vector; front, back, and specular color; front and back transparency; 2D, 3D, and perspective surface texture; and viewport clipping information.

In recent years, demand for high performance graphics systems that can render complex three-dimensional (3D) objects and scenes have increased substantially. This increase is at least in part due to new applications such as computer-generated animation for motion pictures, virtual reality simulators/trainers, and interactive computer games.

These new applications place tremendous demands upon graphics systems. Modern computer displays have also improved and have many more pixels, greater color depth, and are able to display more complex images with higher refresh rates than earlier models. Consequently, modern high performance graphics systems incorporate graphics processors with a great deal of complexity, and the color value of one pixel can be the

accumulated result of many calculations involving several models and mathematical approximations.

With each new generation of graphics system, there is more image data to process, the processing is more complex, and there is less time in which to process it. This need for more processing power is being met with the combination of more hardware resources and more efficient processes.

SUMMARY OF THE INVENTION

One embodiment of the present invention comprises a system and method for performing operations in a graphics system. The graphics system may include an arbitration unit and a plurality of calculation pipelines. Each of the plurality of calculation pipelines is operable to perform a graphics operation. Each of the plurality of calculation pipelines may include digital logic and/or a processing element for performing the graphics operations. The arbitration unit is coupled to each of the plurality of calculation pipelines and is operable to provide graphics operations to selected ones of the plurality of calculation pipelines.

In one embodiment, when assigning operations to respective calculation pipelines, the arbitration unit is operable to determine if a first calculation pipeline is performing a low latency operation. If the arbitration unit determines that the first calculation pipeline is performing a low latency operation, then the arbitration unit provides a next or subsequent operation to the first calculation pipeline. If the arbitration unit determines that the first calculation pipeline is performing a high latency operation, then the arbitration unit provides the next operation to a second calculation pipeline. The second calculation pipeline may be a currently available calculation pipeline, or may be currently performing a low latency operation. In one embodiment, the plurality of calculation pipelines receive operations in a pre-determined manner or pre-determined sequence, and the second calculation pipeline is a next calculation pipeline after the first calculation pipeline in the pre-determined manner or sequence.

In one embodiment, the plurality of calculation pipelines is operable to receive geometric primitive information, such as vertex information, and compute polygons based on the geometric primitive information. For example, the plurality of calculation pipelines may be operable to compute polygon or triangle slopes based on the vertex information.

In one embodiment, a low latency operation comprises an operation that is performable by one of the calculation pipelines in less than a pre-determined number of cycles. A high latency operation may comprise an operation that is performable by one of the calculation pipelines in greater than the pre-determined number of cycles, i.e., a high latency operation comprises an operation that requires greater than the pre-determined number of cycles to be performed by one of the calculation pipelines.

BRIEF DESCRIPTION OF THE DRAWINGS

5 The foregoing, as well as other objects, features, and advantages of this invention may be more completely understood by reference to the following detailed description when read together with the accompanying drawings in which:

Figure 1 is a perspective view of one embodiment of a computer system;

Figure 2 is a simplified block diagram of one embodiment of a computer system;

10 Figure 3 is a functional block diagram of one embodiment of a graphics system;

Figure 4 is a functional block diagram of one embodiment of the media processor of Figure 3;

Figure 5 is a functional block diagram of one embodiment of the hardware accelerator of Figure 3;

15 Figure 6 is a functional block diagram of one embodiment of the video output processor of Figure 3;

Figure 7 is an illustration of a sample space partitioned into an array of bins;

Figure 8 is a functional block diagram of one embodiment of the arbitration unit and calculation pipelines;

20 Figure 9 is a functional block diagram of one embodiment of the pre-setup unit and the five setup units in the render pipeline; and

Figure 10 is a flowchart of one embodiment of an operation allocation method.

25 While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and

detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the present invention as defined by the appended claims. Note, the headings are for organizational purposes only and are not

5 meant to be used to limit or interpret the description or claims. Furthermore, note that the word "may" is used throughout this application in a permissive sense (i.e., having the potential to, being able to), not a mandatory sense (i.e., must)." The term "include", and derivations thereof, mean "including, but not limited to." The term "connected" means "directly or indirectly connected", and the term "coupled" means "directly or indirectly

10 connected".

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Computer System -- Figure 1

5

Figure 1 illustrates one embodiment of a computer system 80 that includes a graphics system. The graphics system may be included in any of various systems such as computer systems, network PCs, Internet appliances, televisions (e.g. HDTV systems and interactive television systems), personal digital assistants (PDAs), virtual reality systems, and other devices that display 2D and/or 3D graphics, among others.

10

As shown, the computer system 80 includes a system unit 82 and a video monitor or display device 84 coupled to the system unit 82. The display device 84 may be any of various types of display monitors or devices (e.g., a CRT, LCD, or gas-plasma display). Various input devices may be connected to the computer system, including a keyboard 86 and/or a mouse 88, or other input device (e.g., a trackball, digitizer, tablet, six-degree of freedom input device, head tracker, eye tracker, data glove, or body sensors). Application software may be executed by the computer system 80 to display graphical objects on display device 84.

15

20 Computer System Block Diagram -- Figure 2

Figure 2 is a simplified block diagram illustrating the computer system of Figure 1. As shown, the computer system 80 includes a central processing unit (CPU) 102 coupled to a high-speed memory bus or system bus 104 also referred to as the host bus 104. A system memory 106 (also referred to herein as main memory) may also be coupled to high-speed bus 104.

25

Host processor 102 may include one or more processors of varying types, e.g., microprocessors, multi-processors and CPUs. The system memory 106 may include any

combination of different types of memory subsystems such as random access memories (e.g., static random access memories or “SRAMs,” synchronous dynamic random access memories or “SDRAMs,” and Rambus dynamic random access memories or “RDRAMs,” among others), read-only memories, and mass storage devices. The system bus or host bus 104 may include one or more communication or host computer buses (for communication between host processors, CPUs, and memory subsystems) as well as specialized subsystem buses.

In Figure 2, a graphics system 112 is coupled to the high-speed memory bus 104. The graphics system 112 may be coupled to the bus 104 by, for example, a crossbar switch or other bus connectivity logic. It is assumed that various other peripheral devices, or other buses, may be connected to the high-speed memory bus 104. It is noted that the graphics system 112 may be coupled to one or more of the buses in computer system 80 and/or may be coupled to various types of buses. In addition, the graphics system 112 may be coupled to a communication port and thereby directly receive graphics data from an external source, e.g., the Internet or a network. As shown in the figure, one or more display devices 84 may be connected to the graphics system 112.

Host CPU 102 may transfer information to and from the graphics system 112 according to a programmed input/output (I/O) protocol over host bus 104. Alternately, graphics system 112 may access system memory 106 according to a direct memory access (DMA) protocol or through intelligent bus mastering.

A graphics application program conforming to an application programming interface (API) such as OpenGL® or Java 3D™ may execute on host CPU 102 and generate commands and graphics data that define geometric primitives such as polygons for output on display device 84. Host processor 102 may transfer the graphics data to system memory 106. Thereafter, the host processor 102 may operate to transfer the graphics data to the graphics system 112 over the host bus 104. In another embodiment, the graphics system 112 may read in geometry data arrays over the host bus 104 using DMA access cycles. In yet another embodiment, the graphics system 112 may be coupled to the system memory 106 through a direct port, such as the Advanced Graphics Port (AGP) promulgated by Intel Corporation.

The graphics system may receive graphics data from any of various sources, including host CPU 102 and/or system memory 106, other memory, or from an external source such as a network (e.g. the Internet), or from a broadcast medium, e.g., television, or from other sources.

Note while graphics system 112 is depicted as part of computer system 80, graphics system 112 may also be configured as a stand-alone device (e.g., with its own built-in display). Graphics system 112 may also be configured as a single chip device or as part of a system-on-a-chip or a multi-chip module. Additionally, in some embodiments, certain of the processing operations performed by elements of the illustrated graphics system 112 may be implemented in software.

Graphics System -- Figure 3

Figure 3 is a functional block diagram illustrating one embodiment of graphics system 112. Note that many other embodiments of graphics system 112 are possible and contemplated. Graphics system 112 may include one or more media processors 14, one or more hardware accelerators 18, one or more texture buffers 20, one or more frame buffers 22, and one or more video output processors 24. Graphics system 112 may also include one or more output devices such as digital-to-analog converters (DACs) 26, video encoders 28, flat-panel-display drivers (not shown), and/or video projectors (not shown). Media processor 14 and/or hardware accelerator 18 may include any suitable type of high performance processor (e.g., specialized graphics processors or calculation units, multimedia processors, DSPs, or general purpose processors).

In some embodiments, one or more of these components may be removed. For example, the texture buffer may not be included in an embodiment that does not provide texture mapping. In other embodiments, all or part of the functionality incorporated in either or both of the media processor or the hardware accelerator may be implemented in software.

In one set of embodiments, media processor 14 is one integrated circuit and hardware accelerator is another integrated circuit. In other embodiments, media

processor 14 and hardware accelerator 18 may be incorporated within the same integrated circuit. In some embodiments, portions of media processor 14 and/or hardware accelerator 18 may be included in separate integrated circuits.

As shown, graphics system 112 may include an interface to a host bus such as host bus 104 in Figure 2 to enable graphics system 112 to communicate with a host system such as computer system 80. More particularly, host bus 104 may allow a host processor to send commands to the graphics system 112. In one embodiment, host bus 104 may be a bi-directional bus.

Media Processor -- Figure 4

Figure 4 shows one embodiment of media processor 14. As shown, media processor 14 may operate as the interface between graphics system 112 and computer system 80 by controlling the transfer of data between computer system 80 and graphics system 112. In some embodiments, media processor 14 may also be configured to perform transformations, lighting, and/or other general-purpose processing operations on graphics data.

Transformation refers to the spatial manipulation of objects (or portions of objects) and includes translation, scaling (e.g. stretching or shrinking), rotation, reflection, or combinations thereof. More generally, transformation may include linear mappings (e.g. matrix multiplications), nonlinear mappings, and combinations thereof.

Lighting refers to calculating the illumination of the objects within the displayed image to determine what color values and/or brightness values each individual object will have. Depending upon the shading algorithm being used (e.g., constant, Gourand, or Phong), lighting may be evaluated at a number of different spatial locations.

As illustrated, media processor 14 may be configured to receive graphics data via host interface 11. A graphics queue 148 may be included in media processor 14 to buffer a stream of data received via the accelerated port of host interface 11. The received graphics data may include one or more graphics primitives. As used herein, the term

graphics primitive may include polygons, parametric surfaces, splines, NURBS (non-uniform rational B-splines), sub-divisions surfaces, fractals, volume primitives, voxels (i.e., three-dimensional pixels), and particle systems. In one embodiment, media processor 14 may also include a geometry data preprocessor 150 and one or more microprocessor units (MPUs) 152. MPUs 152 may be configured to perform vertex transformation, lighting calculations and other programmable functions, and to send the results to hardware accelerator 18. MPUs 152 may also have read/write access to texels (i.e. the smallest addressable unit of a texture map) and pixels in the hardware accelerator 18. Geometry data preprocessor 150 may be configured to decompress geometry, to convert and format vertex data, to dispatch vertices and instructions to the MPUs 152, and to send vertex and attribute tags or register data to hardware accelerator 18.

As shown, media processor 14 may have other possible interfaces, including an interface to one or more memories. For example, as shown, media processor 14 may include direct Rambus interface 156 to a direct Rambus DRAM (DRDRAM) 16. A memory such as DRDRAM 16 may be used for program and/or data storage for MPUs 152. DRDRAM 16 may also be used to store display lists and/or vertex texture maps.

Media processor 14 may also include interfaces to other functional components of graphics system 112. For example, media processor 14 may have an interface to another specialized processor such as hardware accelerator 18. In the illustrated embodiment, controller 160 includes an accelerated port path that allows media processor 14 to control hardware accelerator 18. Media processor 14 may also include a direct interface such as bus interface unit (BIU) 154. Bus interface unit 154 provides a path to memory 16 and a path to hardware accelerator 18 and video output processor 24 via controller 160.

Hardware Accelerator -- Figure 5

One or more hardware accelerators 18 may be configured to receive graphics instructions and data from media processor 14 and to perform a number of functions on the received data according to the received instructions. For example, hardware accelerator 18 may be configured to perform rasterization, 2D and/or 3D texturing, pixel

transfers, imaging, fragment processing, clipping, depth cueing, transparency processing, set-up, and/or screen space rendering of various graphics primitives occurring within the graphics data.

Clipping refers to the elimination of graphics primitives or portions of graphics primitives that lie outside of a 3D view volume in world space. The 3D view volume may represent that portion of world space that is visible to a virtual observer (or virtual camera) situated in world space. For example, the view volume may be a solid truncated pyramid generated by a 2D view window, a viewpoint located in world space, a front clipping plane and a back clipping plane. The viewpoint may represent the world space location of the virtual observer. In most cases, primitives or portions of primitives that lie outside the 3D view volume are not currently visible and may be eliminated from further processing. Primitives or portions of primitives that lie inside the 3D view volume are candidates for projection onto the 2D view window.

Primitives are mapped to a three-dimensional viewport. This involves translating and transforming the objects from their original "world-coordinate" system to the established viewport's coordinates. This creates the correct perspective for three-dimensional objects displayed on the screen.

Screen-space rendering refers to the calculations performed to generate the data used to form each pixel that will be displayed. For example, hardware accelerator 18 may calculate "samples." Samples are points that have color information but no real area. Samples allow hardware accelerator 18 to "super-sample," or calculate more than one sample per pixel. Super-sampling may result in a higher quality image.

Hardware accelerator 18 may also include several interfaces. For example, in the illustrated embodiment, hardware accelerator 18 has four interfaces. Hardware accelerator 18 has an interface 161 (referred to as the "North Interface") to communicate with media processor 14. Hardware accelerator 18 may receive commands and/or data from media processor 14 through interface 161. Additionally, hardware accelerator 18 may include an interface 176 to bus 32. Bus 32 may connect hardware accelerator 18 to boot PROM 30 and/or video output processor 24. Boot PROM 30 may be configured to store system initialization data and/or control code for frame buffer 22. Hardware

accelerator 18 may also include an interface to a texture buffer 20. For example, hardware accelerator 18 may interface to texture buffer 20 using an eight-way interleaved texel bus that allows hardware accelerator 18 to read from and write to texture buffer 20. Hardware accelerator 18 may also interface to a frame buffer 22. For example, hardware
5 accelerator 18 may be configured to read from and/or write to frame buffer 22 using a four-way interleaved pixel bus.

The vertex processor 162 may be configured to use the vertex tags received from the media processor 14 to perform ordered assembly of the vertex data from the MPUs 152. Vertices may be saved in and/or retrieved from a mesh buffer 164.

10 The render pipeline 166 may be configured to rasterize 2D window system primitives and 3D primitives into fragments. A fragment may contain one or more samples. Each sample may contain a vector of color data and perhaps other data such as alpha and control tags. 2D primitives include objects such as dots, fonts, Bresenham lines and 2D polygons. 3D primitives include objects such as smooth and large dots, smooth
15 and wide DDA (Digital Differential Analyzer) lines and 3D polygons (e.g. 3D triangles).

For example, the render pipeline 166 may be configured to receive vertices defining a triangle, to identify fragments that intersect the triangle.

The render pipeline 166 may be configured to handle full-screen size primitives, to calculate plane and edge slopes, and to interpolate data (such as color) down to tile
20 resolution (or fragment resolution) using interpolants or components such as:

r, g, b (i.e., red, green, and blue vertex color);

r2, g2, b2 (i.e., red, green, and blue specular color from lit textures);

alpha (i.e. transparency);

z (i.e. depth); and

25 s, t, r, and w (i.e. texture components).

In embodiments using supersampling, the sample generator 174 may be configured to generate samples from the fragments output by the render pipeline 166 and to determine which samples are inside the rasterization edge. Sample positions may be

defined by user-loadable tables to enable stochastic sample-positioning patterns.

Hardware accelerator 18 may be configured to write textured fragments from 3D primitives to frame buffer 22. The render pipeline 166 may send pixel tiles defining r, s, t and w to the texture address unit 168. The texture address unit 168 may use the r, s, t, and w texture coordinates to compute texel addresses (e.g. addresses for a set of neighboring texels) and to determine interpolation coefficients for the texture filter 170. The texel addresses are used to access data (i.e. texels) from texture buffer 20. The texture buffer 20 may be interleaved to obtain as many neighboring texels as possible in each clock. The texture filter 170 may perform bilinear, trilinear or quadlinear interpolation. The pixel transfer unit 182 may also scale and bias and/or lookup texels. The texture environment 180 may apply texels to samples produced by the sample generator 174. The texture environment 180 may also be used to perform geometric transformations on images (e.g., bilinear scale, rotate, flip) as well as to perform other image filtering operations on texture buffer image data (e.g., bicubic scale and convolutions).

In the illustrated embodiment, the pixel transfer MUX 178 controls the input to the pixel transfer unit 182. The pixel transfer unit 182 may selectively unpack pixel data received via north interface 161, select channels from either the frame buffer 22 or the texture buffer 20, or select data received from the texture filter 170 or sample filter 172.

The pixel transfer unit 182 may be used to perform scale, bias, and/or color matrix operations, color lookup operations, histogram operations, accumulation operations, normalization operations, and/or min/max functions. Depending on the source of (and operations performed on) the processed data, the pixel transfer unit 182 may output the processed data to the texture buffer 20 (via the texture buffer MUX 186), the frame buffer 22 (via the texture environment unit 180 and the fragment processor 184), or to the host (via north interface 161). For example, in one embodiment, when the pixel transfer unit 182 receives pixel data from the host via the pixel transfer MUX 178, the pixel transfer unit 182 may be used to perform a scale and bias or color matrix operation, followed by a color lookup or histogram operation, followed by a min/max function. The pixel transfer unit 182 may then output data to either the texture buffer 20 or the frame buffer 22.

Fragment processor 184 may be used to perform standard fragment processing operations such as the OpenGL® fragment processing operations. For example, the fragment processor 184 may be configured to perform the following operations: fog, area pattern, scissor, alpha/color test, ownership test (WID), stencil test, depth test, alpha blends or logic ops (ROP), plane masking, buffer selection, pick hit/occlusion detection, and/or auxiliary clipping in order to accelerate overlapping windows.

Texture Buffer 20

Texture buffer 20 may include several SDRAMs. Texture buffer 20 may be configured to store texture maps, image processing buffers, and accumulation buffers for hardware accelerator 18. Texture buffer 20 may have many different capacities (e.g., depending on the type of SDRAM included in texture buffer 20). In some embodiments, each pair of SDRAMs may be independently row and column addressable.

Frame Buffer 22

Graphics system 112 may also include a frame buffer 22. In one embodiment, frame buffer 22 may include multiple memory devices such as the 3D-RAM memory devices manufactured by Mitsubishi Electric Corporation. Frame buffer 22 may be configured as a display pixel buffer, an offscreen pixel buffer, and/or a supersample buffer. Furthermore, in one embodiment, certain portions of frame buffer 22 may be used as a display pixel buffer, while other portions may be used as an offscreen pixel buffer and sample buffer.

Video Output Processor -- Figure 6

A video output processor 24 may also be included within graphics system 112. Video output processor 24 may buffer and process pixels output from frame buffer 22. For example, video output processor 24 may be configured to read bursts of pixels from

frame buffer 22. Video output processor 24 may also be configured to perform double buffer selection (dbsel) if the frame buffer 22 is double-buffered, overlay transparency (using transparency/overlay unit 190), plane group extraction, gamma correction, psuedocolor or color lookup or bypass, and/or cursor generation. For example, in the
5 illustrated embodiment, the output processor 24 includes WID (Window ID) lookup tables (WLUTs) 192 and gamma and color map lookup tables (GLUTs, CLUTs) 194. In one embodiment, frame buffer 22 may include multiple 3DRAM64s 201 that include the transparency overlay 190 and all or some of the WLUTs 192. Video output processor 24 may also be configured to support two video output streams to two displays using the two
10 independent video raster timing generators 196. For example, one raster (e.g., 196A) may drive a 1280x1024 CRT while the other (e.g., 196B) may drive a NTSC or PAL device with encoded television video.

DAC 26 may operate as the final output stage of graphics system 112. The DAC 26 translates the digital pixel data received from GLUT/CLUTs/Cursor unit 194 into
15 analog video signals that are then sent to a display device. In one embodiment, DAC 26 may be bypassed or omitted completely in order to output digital pixel data in lieu of analog video signals. This may be useful when a display device is based on a digital technology (e.g., an LCD-type display or a digital micro-mirror display).

DAC 26 may be a red-green-blue digital-to-analog converter configured to
20 provide an analog video output to a display device such as a cathode ray tube (CRT) monitor. In one embodiment, DAC 26 may be configured to provide a high resolution RGB analog video output at dot rates of 240 MHz. Similarly, encoder 28 may be configured to supply an encoded video signal to a display. For example, encoder 28 may provide encoded NTSC or PAL video to an S-Video or composite video television
25 monitor or recording device.

In other embodiments, the video output processor 24 may output pixel data to other combinations of displays. For example, by outputting pixel data to two DACs 26 (instead of one DAC 26 and one encoder 28), video output processor 24 may drive two CRTs. Alternately, by using two encoders 28, video output processor 24 may supply
30 appropriate video input to two television monitors. Generally, many different

combinations of display devices may be supported by supplying the proper output device and/or converter for that display device.

Sample-to-Pixel Processing Flow

5

In one set of embodiments, hardware accelerator 18 may receive geometric parameters defining primitives such as triangles from media processor 14, and render the primitives in terms of samples. The samples may be stored in a sample storage area (also referred to as the sample buffer) of frame buffer 22. The samples are then read from the sample storage area of frame buffer 22 and filtered by sample filter 22 to generate pixels. The pixels are stored in a pixel storage area of frame buffer 22. The pixel storage area may be double-buffered. Video output processor 24 reads the pixels from the pixel storage area of frame buffer 22 and generates a video stream from the pixels. The video stream may be provided to one or more display devices (e.g. monitors, projectors, head-mounted displays, and so forth) through DAC 26 and/or video encoder 28.

The samples are computed at positions in a two-dimensional sample space (also referred to as rendering space). The sample space may be partitioned into an array of bins (also referred to herein as fragments). The storage of samples in the sample storage area of frame buffer 22 may be organized according to bins as illustrated in Figure 7. Each bin may contain one or more samples. The number of samples per bin may be a programmable parameter.

Arbitration Unit and Calculation Pipelines -- Figure 8

One embodiment of the present invention is illustrated in Figure 8 and comprises a system for performing operations in a graphics system. The graphics system may include an arbitration unit 810 and a plurality of calculation pipelines 820 - 828. Each of the plurality of calculation pipelines 820 - 828 is operable to perform a graphics operation. Each of the plurality of calculation pipelines 820 - 828 may include digital logic and/or a processing element for performing the graphics operations. The arbitration

unit 810 is coupled to each of the plurality of calculation pipelines 820 - 828 and is operable to provide graphics operations to selected ones of the plurality of calculation pipelines 820 - 828. Another embodiment of the present invention may have two pipelines. Still another embodiment of the present invention may have three or more pipelines.

Pre-setup and Setup Units -- Figure 9

Another embodiment of the present invention is a portion of the render pipeline 166 that is illustrated in the block diagram in Figure 9. The first module of the render pipeline 166 is the pre-setup unit 910. The second module of the render pipeline 166 contains 5 setup units 920 - 928 that function in parallel. The render pipeline 166 may be configured to rasterize 2D window system primitives and 3D primitives into fragments.

Operation Allocation Method -- Figure 10

Figure 10 illustrates one embodiment of a method of allocating a next operation to one of a set of calculation pipelines such as calculation pipelines 820 - 828. The arbitration unit 810 is operable to determine the duration of an operation of a first calculation pipeline such as calculation pipeline 824 (step 1000). If the operation is less than a threshold value, the arbitration unit 810 determines the operation to be of a short duration and is referred to herein as a low latency operation (step 1010). The arbitration unit 810 then provides the next (or subsequent) operation to the first calculation pipeline 824 (step 1020). If the arbitration unit 810 determines that the first calculation pipeline 824 is performing an operation of a long duration (an operation of duration greater than a threshold value referred to herein as a high latency operation), then the arbitration unit 810 sends the next operation to a second calculation pipeline such as calculation pipeline 826 (step 1030). In some embodiments, the second calculation pipeline may be any one of the calculation pipelines 820 - 828 that is currently available, or any one that may be currently performing a low latency operation. In one embodiment, the plurality of calculation pipelines 820 - 828 receive operations in a pre-determined manner or pre-

determined sequence, and the second calculation pipeline is a next calculation pipeline after the first calculation pipeline in the pre-determined manner or sequence. In some embodiments, the sequence is a repeating sequence of calculation pipelines A, B, C, D, and E.

5 In one embodiment, a low latency operation comprises an operation that is performable by one of the calculation pipelines 820 – 828 in less than a first pre-determined number of clock cycles. A high latency operation may comprise an operation that is performable by one of the calculation pipelines 820 – 828 in greater than a second pre-determined number of clock cycles, i.e., a high latency operation comprises an
10 operation that requires greater than the pre-determined number of clock cycles to be performed by one of the calculation pipelines 820 – 828. In some embodiments, the first pre-determined number of clock cycles and the second pre-determined number of clock cycles are the same. In some embodiments, the pre-determined number of clock cycles is two clock cycles for both the high and the low latency threshold comparison. In some
15 embodiments, the clock frequency is 166 megahertz.

In one embodiment, the plurality of setup units 920 – 928 are operable to receive geometric primitive information, such as vertex information, from the pre-setup unit 910 and compute values for the polygons corresponding to the geometric primitive information. For example, the plurality of setup units 920 – 928 may be operable to
20 compute polygon or triangle edge slopes based on the vertex information.

Although the embodiments above have been described in considerable detail, other versions are possible. Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.

25 Note the section headings used herein are for organizational purposes only and are not meant to limit the description provided herein or the claims attached hereto.